# **Einführung in Algorithmen**

"An algorithm is the most powerful weapon you can use to tame the complexity that underlies almost everything." **Robert Sedgewick**

Willkommen im Modul zu Algorithmen und Datenstrukturen! Im ersten Teil des Moduls werden wir uns mit Algorithmen beschäftigen. Sie sind essentielle Bausteine der Informatik, die es uns ermöglichen, komplexe Probleme systematisch zu lösen. Im Anschluss daran werden wir das Thema Datenstrukturen in den Fokus rücken. Datenstrukturen sind eng mit Algorithmen verbunden und spielen eine entscheidende Rolle bei der Speicherung und Organisation von Daten. Gemeinsam werden wir erkunden, wie diese beiden zentralen Konzepte zusammenhängen und wieso es essentiell für Programmierer:innen ist, sie zu verstehen.

Algorithmen sind allgegenwärtig: Suchmaschinen nutzen sie, um uns möglichst passende Ergebnisse anzuzeigen, Navigationsapps, um den schnellsten Weg zum Ziel zu ermitteln, Social Media, um uns Einträge zu zeigen, die unseren Interessen entsprechen oder uns passende Kontakte vorzuschlagen. Der Begriff aus der Informatik hat es inzwischen in den allgemeinen Sprachgebrauch geschafft. Aber was genau meint der Begriff Algorithmus?

## Ein Algorithmus ist eine klar definierte Handlungsanweisung

Ein Algorithmus ist, grob gesagt, eine Handlungsvorschrift zur Lösung eines Problems. Sie kann als Computerprogramm implementiert werden oder in menschlicher Sprache verfasst sein, und von Menschen ausgeführt werden. Ihre Ausführung sollte korrekte Ergebnisse liefern. Wir sehen hier von einer rigorosen mathematischen Definition ab, die recht abstrakt ist. Damit wir eine Handlungsanweisung als Algorithmus bezeichnen dürfen, sollte sie aber mindestens eine Reihe von Kriterien erfüllen:

* Die Schritte der Anweisung sind in einem **endlichen Text** eindeutig beschreibbar.
* Jeder Schritt musstatsächlich **ausführbar sein**
* Die Anweisung darf nur **endlich viele Schritte** zur Ausführung benötigen.
* Die Anweisung muss mit **endlich viel Speicherplatz** auskommen.

Schauen wir uns ein Beispiel an: Angenommen, wir möchten in einer unsortierten Bibliothek mit einem einzigen Regal einen bestimmten Buchtitel finden. In der Bibliothek sind die Bücher linksbündig angeordnet, und es sind keine Lücken zwischen den Büchern vorhanden. Wir könnten folgendermaßen vorgehen:

1. Beginne ganz links im Regal mit der Suche.
2. Wenn kein Buch an der Position vorhanden ist, beende die Suche und melde, dass der gesuchte Titel nicht gefunden wurde.
3. Wenn an der Position ein Buch steht, schaue dir den Titel des Buches an. Wenn der Titel dem gesuchten entspricht, gib das Buch und dessen Position aus und beende die Suche.
4. Bewege dich eine Position nach rechts und wiederhole alle Schritte ab Schritt 2.

Wir betrachten hier eine Handlungsvorschrift, die, sofern vorhanden, uns den korrekten Buchtitel ausgibt. Erfüllt sie die oben genannten Kriterien?

Der **beschreibende Text ist offensichtlich endlich**, denn er ist durch die Schritte 1) - 4) beschreibbar. Jeder **Schritt ist ausführbar** und könnte von einer anderen Person umgesetzt werden**.** Der benötigte **Speicherplatz ist endlich.** Eine Personmuss sich lediglich merken, an welcher Position sie grade sucht, und welches der gesuchte Titel ist. Sie muss sich nicht einmal merken, welche Titel sie bereits gesehen hat. Der benötigte Speicherplatz ist im Vergleich zu vielen anderen Aufgaben also sogar sehr begrenzt. Würde der Algorithmus von einem Computer ausgeführt werden, würde `begrenzter Speicherplatz` analog hierzu bedeuten, dass er mit einer mit einer festgelegten Menge an Ressourcen, also Speicherplatz durch den verwendeten Arbeitsspeicher oder andere Speicherressourcen, abgeschlossen werden kann. Zuletzt bedarf der Prozess nur **endlich vieler Schritte**: Die Suche wird nie unendlich lange dauern. Es gibt eine endliche Anzahl von Büchern im Regal. Sobald alle Bücher überprüft wurden, oder das gesuchte Buch gefunden wurde, endet der Prozess.

Unsere Handlungsanweisung erfüllt also die Mindestkriterien für Algorithmen. Nun bist du dran. Beantworte die folgende Aufgabe, indem du eine Handlungsanweisung suchst, die mindestens eins der Kriterien nicht erfüllt. In der folgenden aufklappbaren Box findest du einen möglichen Lösungsvorschlag.

Quiz: Beschreibe die Suche nach einem Buchtitel in einer nicht-algorithmischen Weise.

Antwort (aufklappbare Box): Eine Möglichkeit wäre die folgende:

1. Nimm dir zufällig ein Buch aus dem Regal
2. Wenn der Titel dem gesuchten entspricht, gib das Buch aus und beende die Suche.
3. Ansonsten wiederhole die Schritte beginnend mit 1)

Die Ausführung dieser Handlungsvorschrift hat potenziell unendlich viele Schritte. Wird das Buch nicht gefunden, so beendet kein Mechanismus den Prozess. Er ist potenziell sogar dann unendlich, wenn der Titel im Regal vorhanden ist. Hast du eine andere Lösung gefunden? Dann poste sie doch gerne im Forum!

## Für die Effizienzmessung benötigen wir ein objektives Maß

Unser Beispielalgorithmus liefert uns korrekte Ergebnisse und erfüllt die Mindestkriterien für Algorithmen. Eine wichtige Frage sind wir bei unseren Betrachtungen allerdings noch nicht angegangen: Ist der Algorithmus der bestmögliche? Daran schließt sich die nächste Frage an: Was macht einen Algorithmus besser als einen anderen? Dass ein Algorithmus schnell Ergebnisse produziert, ist auf jeden Fall eine wünschenswerte Eigenschaft. Wir sprechen dann von der \*Laufzeit\* eines Algorithmus.

Aber warum sollten wir uns um die Laufzeit Gedanken machen? Sind Computer heutzutage nicht so schnell, dass der Weg zum Ziel nachrangig ist? Nicht ganz, denn auch die Menge an Daten, die verarbeitet wird, wird immer größer.

Im Jahr 2021 gab es etwa 130 Billionen (130.000.000.000.000) Websites im Internet. Stellen wir uns einmal vor, Googles Algorithmus würde bei jeder Suchanfrage jede dieser Websites nacheinander durchforsten. Würde die Überprüfung einer einzelnen Seite auch nur ein Hundertstel einer Millisekunde benötigen, müssten wir mehr als 41 Jahre auf unser Ergebnis warten! Ein solcher Algorithmus wäre vollkommen nutzlos. Es ist nicht nur wichtig, dass ein Algorithmus ein Problem löst, er muss es auch in akzeptabler Zeit tun.

Vertiefung: **Zwei Facetten der Effizienz: Laufzeit und Speicherplatzbedarf.**

Sprechen wir von der Effizienz eines Algorithmus, beziehen wir uns in der Regel auf seine Laufzeit. Aber da gibt es noch etwas: wieviel Speicher benötigt er zur Ausführung? Nehmen wir unser Buchsuchbeispiel: Müsste sich die ausführende Person nicht nur die aktuelle Position, sondern auch die bereits angesehenen Titel merken, würde ihr Gedächtnis stärker beansprucht werden. Anders ausgedrückt: Der Speicherplatzbedarf würde steigen. Ein solcher Algorithmus hätte eine geringere Speicherplatzeffizienz.

Bei der Bewertung von Algorithmen liegt der Fokus in den allermeisten Anwendungen auf der Laufzeiteffizienz. Das liegt daran, dass in vielen Anwendungen die Geschwindigkeit der Ausführung entscheidend ist, insbesondere in Echtzeitsystemen oder bei großen Datenmengen. In Umgebungen mit begrenzten Ressourcen oder mit signifikanten Speicherkosten ist die Speicherplatzeffizienz jedoch ein wichtiger Faktor.

Wir wollen die Laufzeit eines Algorithmus quantifizieren. Wie können wir das tun?

Betrachten wir nochmal unseren Buchsuchalgorithmus. Wir könnten experimentell vorgehen und die Zeit messen. Dafür könnten wir in vielen unterschiedlich großen Bibliotheken viele verschiedene Titel suchen, und messen, wie lange die Suche jeweils dauert. Wir könnten dann versuchen, einen Durchschnittswert zu ermitteln, möglicherweise in Abhängigkeit der Größe der Bibliothek.

Allerdings gibt es bei einem solchen Ansatz Unsicherheiten hinsichtlich der Skalierbarkeit. Ein weiteres Beispiel ist die Konsistenz: Möglicherweise haben wir in der ersten Hälfte des Regals noch Motivation und gehen schnell vor, lassen uns in der zweiten Hälfte aber häufiger ablenken. Eine andere Person sieht hingegen vielleicht in der zweiten Hälfte das Ende nahen und arbeitet die Bücher schneller ab. Das ist ein Problem. Diese Variabilität beeinträchtigt die Vergleichbarkeit von Algorithmen, da das, was für uns gut funktioniert, nicht unbedingt für andere Personen ähnlich gut funktioniert.

Implementieren wir Algorithmen in Computerprogrammen, können wir analog für eine Reihe von Beispielen die Zeit messen, die unser Programm zur Ausführung benötigt. In Python ist das mit dem Modul `timeit` möglich. Das Kernproblem ist aber auch dasselbe: Die Vergleichbarkeit ist nicht gegeben. Verschiedene Rechner haben verschiedene Prozessoren, die für verschiedene Aufgaben optimiert wurden. Dadurch erledigen sie manche Aufgaben schneller und andere langsamer.

Objektiver ist, die **Anzahl der Schritte** zu zählen, die wir, oder unsere Computerprogramme, benötigen, bis der Prozess stoppt. Ein Schritt ist eine einzelne Operation, die ausgeführt wird. Wie viele Operationen benötigt werden, ist vollkommen unabhängig von der ausführenden Person oder dem Computer.

Wollen wir die eine allgemeine Bewertung anhand der benötigten Operationen vornehmen, stellt uns das allerdings vor weitere Schwierigkeiten. In unserem Buchsuchbeispiel könnten wir beispielsweise jedem möglichen Buchtitel eine Wahrscheinlichkeit zuordnen, mit der er gesucht wird, und anschließend die Anzahl der benötigten Schritte pro Titel mit dieser Wahrscheinlichkeit gewichten.

Ein solcher Ansatz ist unnötig aufwendig, denn er gibt eine sehr genaue Einschätzung der Laufzeit. Wir sind aber vor allem daran interessiert, zu erfahren, wie gut Algorithmen auf sehr großen Datensätzen funktionieren. Um genau zu sein, interessiert uns noch nicht einmal unbedingt, wie gut ein Algorithmus funktioniert, sondern eher, wie schlecht er im schlechtesten Fall funktioniert. Wie lange müssen wir maximal auf eine Google-Suchanfrage warten?

Durchgesetzt hat sich daher ein anderer Ansatz: die Betrachtung der **asymptotischen Laufzeit** für den "worst case". Dies gibt uns eine obere Grenze dafür, wie lange der Algorithmus in den schwierigsten Fällen brauchen würde. Es hilft uns, Algorithmen in einer Weise zu bewerten, die unabhängig von spezifischen Inputs oder subjektiven Gewichtungen ist. Die asymptotische Betrachtung der Laufzeit nennen wir *Laufzeitkomplexität*.

Die Schreibweise für das Grenzwertverhalten eines Algorithmus im schlechtesten Fall ist die *Big O Notation*. In der folgenden Übung werden wir sie genauer kennenlernen.

Merke:

* Endlicher Text, Ausführbarkeit, Endlich viele Schritte und endlich viel Speicherplatz sind Mindestanforderungen an eine Handlungsanweisung, damit sie ein Algorithmus sein kann.
* Die asymptotische Betrachtung der benötigten Schritte nennen wir Laufzeitkomplexität.
* Die Big O Notation ist die Schreibweise für die Laufzeitkomplexität im schlechtesten Fall.